Final Project Report (Whack-a-Mole)

Team 6 (Atmadiya Debnath, Moe Sakamoto, Nolan Lewis, Qing Wen, Soumya Bandyopadhyay)
May 2, 2022

Abstract

We targeted designing an autonomous robot by integrating sensors and actuators to a retrofitted mechanical chassis to ensure that it can traverse a square field, recognize colors, and drop moles at locations per the choice of the user, in a fixed amount of time. Our team experimented with four-wheel and two-wheel drives and custom-designed a position-controlled servo-driven mole-dispensing system, manufactured out of packaging board. We transitioned from an open loop four-wheel drive, receiving feedback from the Ultrasonic sensor to a PD-control closed-loop two-wheel drive with the ultrasonic sensor, the feedback loop connected only to the line follower. Our robot can successfully move in a straight line and dispense moles, based on the initially selected color, and visual indicators for the chosen color and the states corresponding to the start and end of the task.

1. Introduction

Our objective is to build a robot that can traverse a square field. The target is to place mole-whackers on the squares with moles. The squares occupied by moles are detected by color. The colors chosen for the squares have three colors (red, yellow, and blue), with the starting square as white. At the start of the game, the "Game-Master" would set the color of the square where the moles are hiding. The objective of the whacker is to autonomously search the playing field for the moles and place mole-whackers on those squares. We had two important design iterations: 1. Four-wheel drive and 2. Two Wheel drive. Since the two-wheel drive is our final design, the sections prior to the "Results and Discussion" have focused on this specific design, unless otherwise mentioned.

2. Functional Decomposition to Design

We targeted designing the robot (see Figure 1) within a rectangular bounding box of 12 in. x 12 in. x 12 in and under a cost of $300 (25% lower than the maximum budget). We have selected an efficient drive system of DC motors which take feedback from the sensors (ultrasonic sensors, line follower, and color). The robot receives power from a single battery (9 V) which powers the H bridge (L298N Dual H Bridge) that interfaces with 2 DC motors. The ultrasonic, line follower, and color sensors receive power from the Arduino Mega. The direction of rotation of the motors is controlled using digital signals from the Arduino Mega and the speed of the motors is controlled

by the amplitude of the PWM signals. The chassis of the robot has two compartments. The bottom layer has the battery, the H Bridge, Arduino, and the prototyping board, while the top layer has the buttons and the indicator LEDs. The mole dispensing system comprises a hollow tower with rectangular cross-section, which holds the moles. There is a semi-circular plate with an opening for the mole to drop. The semicircular plate consists of a slot to insert the horn of the servo, the rotation of which controls the dropping of the mole. The interconnection of the electronic hardware is depicted by the circuit diagram in the Appendix.

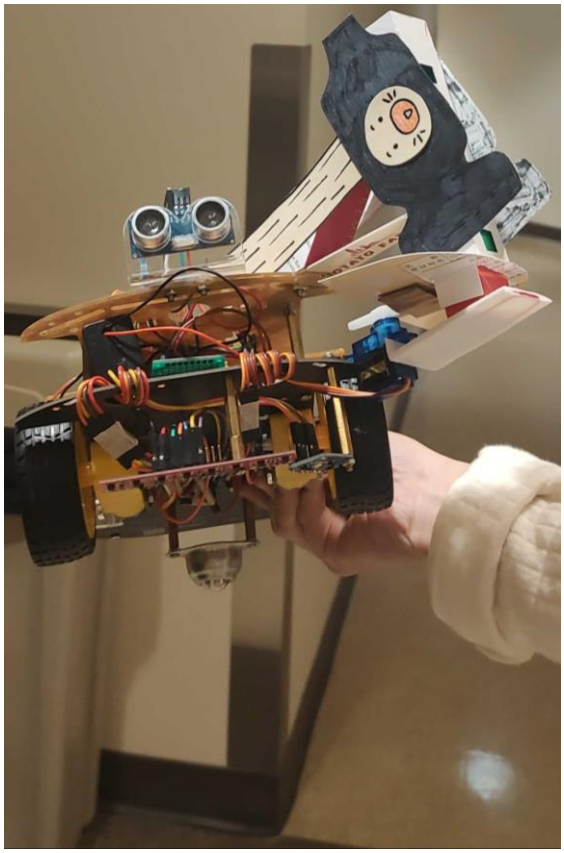


Figure 1: Picture of the Complete Robot

3. Sensor and Actuator Calibration

Figure 2: Testing of the Ultrasonic Sensor, Considering Objects at Different Distances. Top Left: 2 cm; Top Right: 5 cm; Middle Left: 21 cm; Middle Right: 34 cm; Bottom: 170 cm

Ultrasonic Sensor: The ultrasonic sensor used in the project is HC-SR04. It uses SONAR to check the distance from the sensor. We included two modules: ultrasonic transmitter and receiver. It has a high range of non-contact detection with high accuracy and stability. The range of this sensor is 2 cm to 400 cm with an accuracy range of 3 mm. We calibrated the ultrasonic sensor at various distances ranging from 2cm to 170cm. Our maximum range of calibration is till 170 cm. The images for the testing of the ultrasonic sensor and the corresponding recorded values of the distance are included in Fig. 2.

RGB Color Sensor: We used color sensor TCS34725, it has elements to sense RGB and clear light. It has an IR blocking filter which minimizes the spectral component of the incoming light and allows color detection with better accuracy. We determined four colors through the color

sensor which are Red, Yellow, Blue, and White.

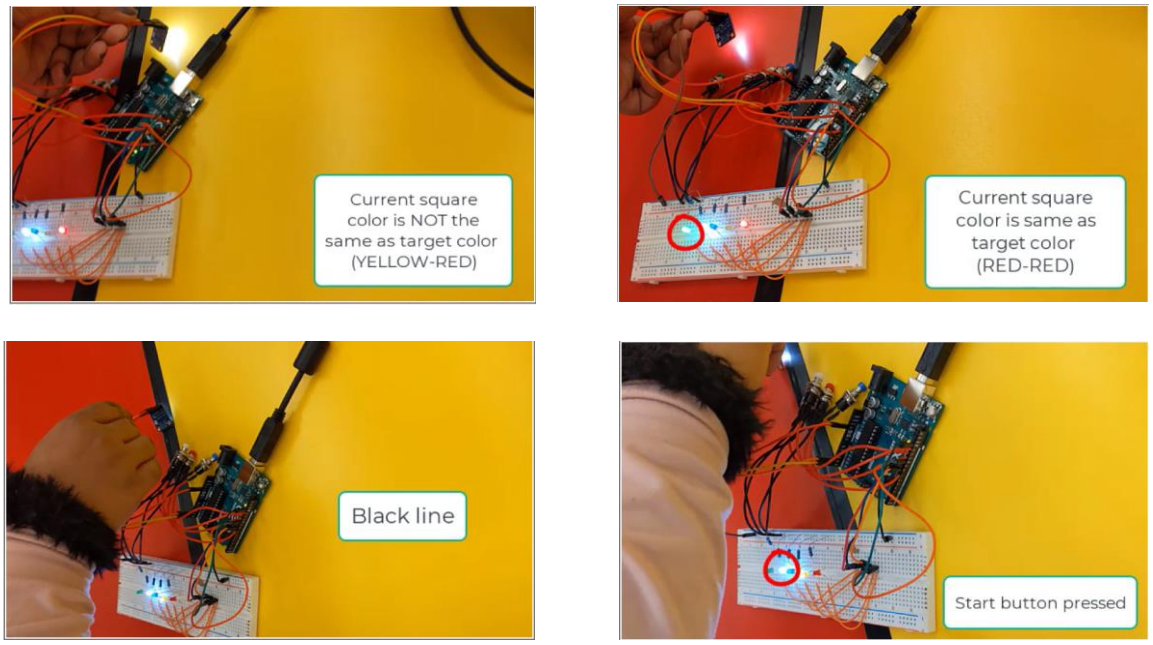


Figure 3: Demonstration of Functionality of Color Sensor

DC Motor Actuator: We used two DC motors (3 – 6 V), to power two wheels. In our robot, we used a single 9 V battery to power two DC motors, so the input voltage to each motor lies at approximately ~ 4.5 V, which lies within the threshold. The DC motors are controlled using the L298N Dual H bridge. The direction of rotation of individual motors is controlled using the digital input pins and the amplitude of the motion is controlled by the amplitude of the PWM signals from the Arduino Mega. Figure 4 depicts the wheels powered by motors (4 Wheel Drive)

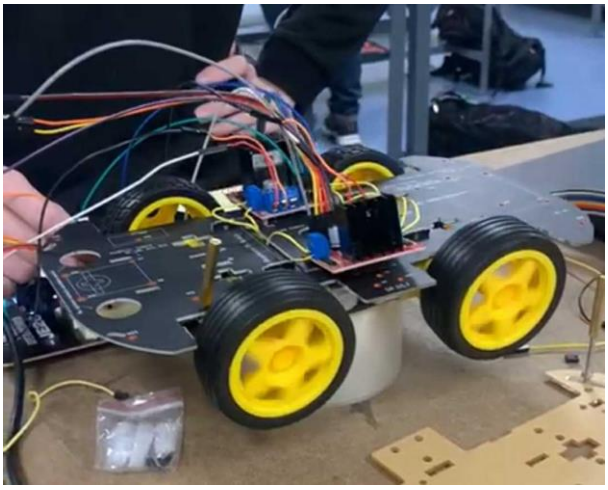


Figure 4: Image of Wheels Powered by DC Motors

Line Follower Sensor: We are using Pololu QTR-8RC reflectance sensor array, which is intended as a line follower sensor. It carries IR emitter and photoresistor receiver pair evenly spaced. For calibration, we exposed each of the reflectance sensors to the lightest and darkest readings. For line follower calibration, we slide it across all the colors at the calibration phase so that each sensor gets the range of readings for our required colors.

4. Motion Planning and Controls

Path Planning

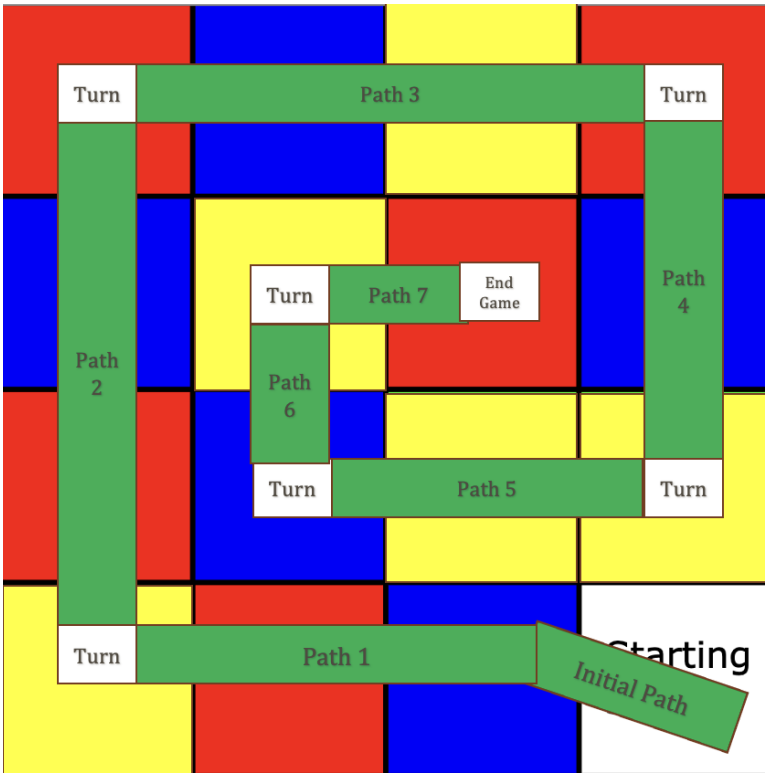


Figure 5: Path Plan

The chosen path relies on the robot to traverse every square resembles a spiral pattern as seen in Figure 5. We selected this path since it limits the robot to only turn right at certain points in the path (signified by the turn box above).

Control Strategy

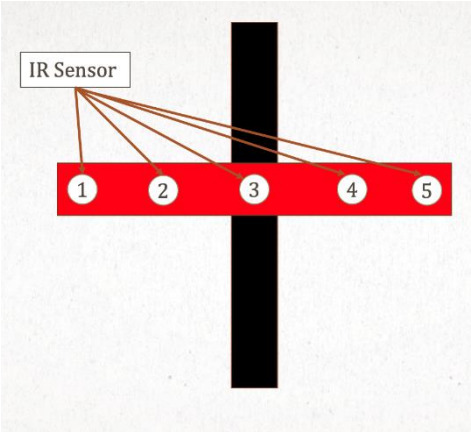


Figure 6: IR Sensor Readings

We envisioned a spiral path for the robot. However, implementing the path in practice is not straightforward initially. A line follower sensor is integrated to ensure that the robot follows the line that resembled the spiral pattern path in Figure 5. For Paths 1-7, a PD controller is employed to ensure the line follower sensor helped follow the black line on the path. At the turn points, open loop control is employed to turn the robot and then return to the logic for PD control when it gets back on the Paths.

In Figure 6, a simplified representation of the line following sensor is used to show the positions and numbers corresponding to each IR/reflectance sensor. A built-in function called readLineBlack() is called in the FSM code which returns an estimate of the position of the black line. The estimate is made using a weighted average of the sensor indices multiplied by 1000, so that a return value of 0 indicates that the line is directly below sensor 1, a return value of 1000 indicates that the line is directly below sensor 2, 2000 indicates that it's below sensor 3, etc. Intermediate values indicate that the line is between two sensors.

$$\frac{(0 \times v_0) + (1000 \times v_1) + (2000 \times v_2) + \cdots}{v_0 + v_1 + v_2 + \cdots}$$

The PD control is determined by keeping setpoint 2000, which corresponds to the 3rd line sensor in Figure 6. The error is subsequently measured and PID control is implemented to ensure the robot follows the line.

Ball Dispensing Strategy

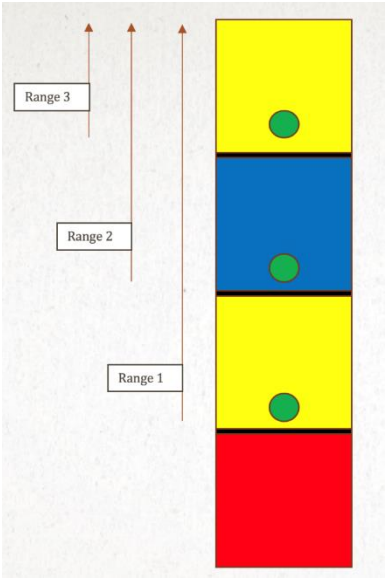


Figure 7: Ball Dispensing Strategy with Distances from Ultrasonic Sensors

Once the path for the robot worked as intended, the logic for dropping the mole whackers is implemented in the code. The position of the robot relative to the board is determined using the ultrasonic sensor. For each path, there are three ranges of distances, detected by the ultrasonic sensor that determine whether the robot is in a specific square. This is used to ensure the ball is not dropped multiple times in the same square.

FSM

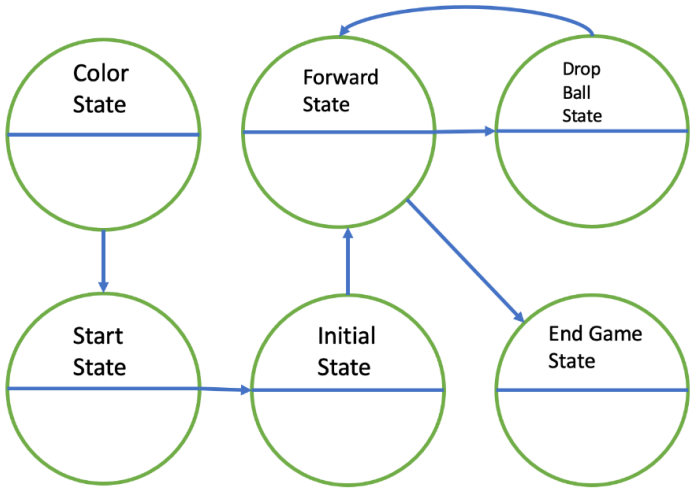


Figure 8: Simplified FSM (Please see Code in Appendix for Implementation)

Inputs and Outputs

Figure 8 reflects a simplified FSM that does not detail the additional functions that were used to control the robot. There are five inputs that are required to be physically initiated by the user. There are three buttons representing the three colors of the grid. There is also another button that controls whether the game starts or not. There is also an emergency stop button that shuts off the motors. In addition, there are also other inputs from other sensors such as the ultrasonic sensor, color sensor, and line follower sensor that are employed for the motion planning of the robot. The inputs are employed in each state to determine the progression of the finite state machine. For example, the color state can only move to the start state if the start button is also triggered. This allows the user to first select the color before starting the game. The forward state which controls the path planning of the robot can only move to the drop ball state or end game state assuming the emergency stop button has not been triggered.

5. Results and Discussion

The robot went through multiple design iterations. Initially, the robot consisted of four wheels driven by four DC motors, and a line following sensor to determine where the robot is on the playing field. The line following sensor would count the number of lines crossed when it passes a horizontal line. And depending on the number of lines crossed, we knew which square the robot is in. However, this method is unreliable. The robot would detect a line even though it is on a vertical line. Due to this outcome, an ultrasonic sensor is incorporated in the design. Based on the distance from the ultrasonic sensor, the robot determines which square it is in, while the line sensor makes sure the robot follows the line. As for the four wheels, it is changed to two DC motor wheels and one dummy wheel when a PD controller is implemented. This decision is made because the built-in function in the line sensor library controls the speed of two wheels instead of four. With these changes, the robot is able to traverse through the whole path and drop the moles in the correct square. However, the motor speed is extremely sensitive to the power supplied by the battery. Thus, during the competition, when the battery is changed, the robot is only able to complete the first straight path and drop one mole.

Overall, the line following method worked well for the straight path with the mole dispensing accurately. The one issue we had is turning. Since the turning control is open loop with no feedback, the turning would only work well when a certain amount of power is supplied from the

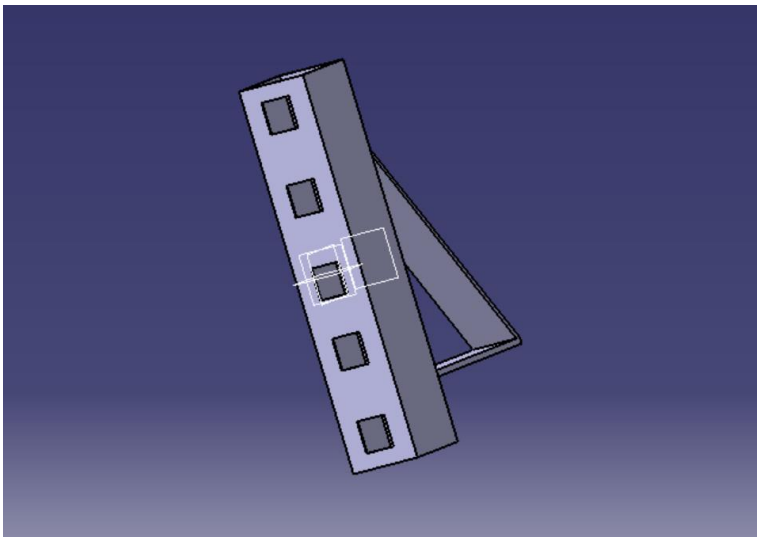
battery. Thus, with the current program logic, the turning and base motor speed has to be calibrated to the current battery power every time before the game. If we do further design iterations of the robot, one major improvement would be to include wheel encoders. This would allow the robot to follow the path and turn accurately without relying on the battery level. Another improvement would be to introduce LiPo batteries with step down DC-DC buck converters towards ensuring a fixed output voltage for the DC motors, over a wide range of battery voltage levels. This, in turn, would entail a fixed input voltage to the DC motors, thereby enhancing the efficacy and robustness of the software-based PD control.

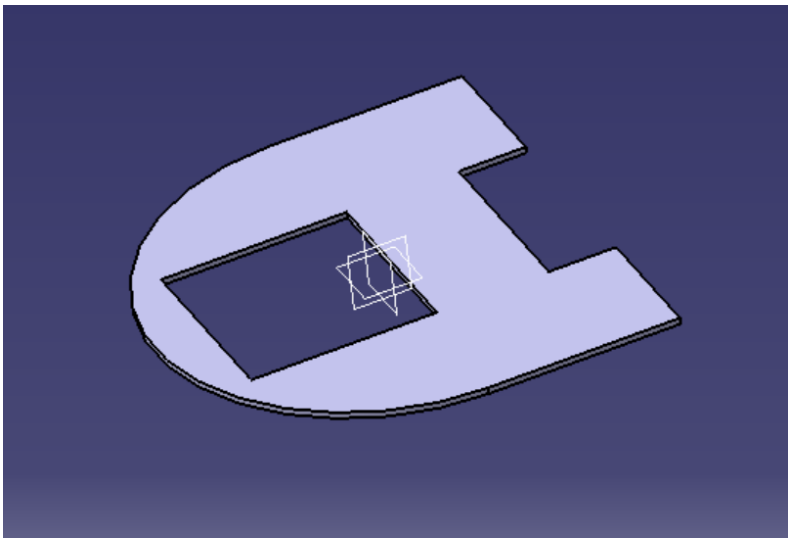
Conclusion

In conclusion, we designed an autonomous robot through an iterative process, driven by the integration of the control logic with the procured hardware and designed chassis for achieving the desired functionalities. We started with a four-wheel drive for the robot, driven by four DC motors, controlled in open loop, by taking feedback from the line follower and ultrasonic sensors. The mole-dispensing system is designed in CAD and manufactured out of packaging board. This system is actuated by controlling the position of the micro-servo, based on the feedback received from the color sensor. We encountered major hurdles, in intercepting the feedback from the line follower because of faulty hardware and variation in illumination conditions during testing. To address these drawbacks, we transitioned to a two-wheel drive, while seeking combined feedback from the line follower and the ultrasonic sensor for navigating across individual squares. To minimize the complexity of the software, we switched from an open loop to a PD controller, based on the feedback from the line-follower. While the turning of the robot is not demonstrated, it could be potentially implemented with a few iterations with the parameters for PD control and corresponding time delays in the code. The designed robot is able to successfully move in a straight line and dispense moles, based on the initially selected color, and visual indicators for the chosen color and the states corresponding to the start and end of the task.

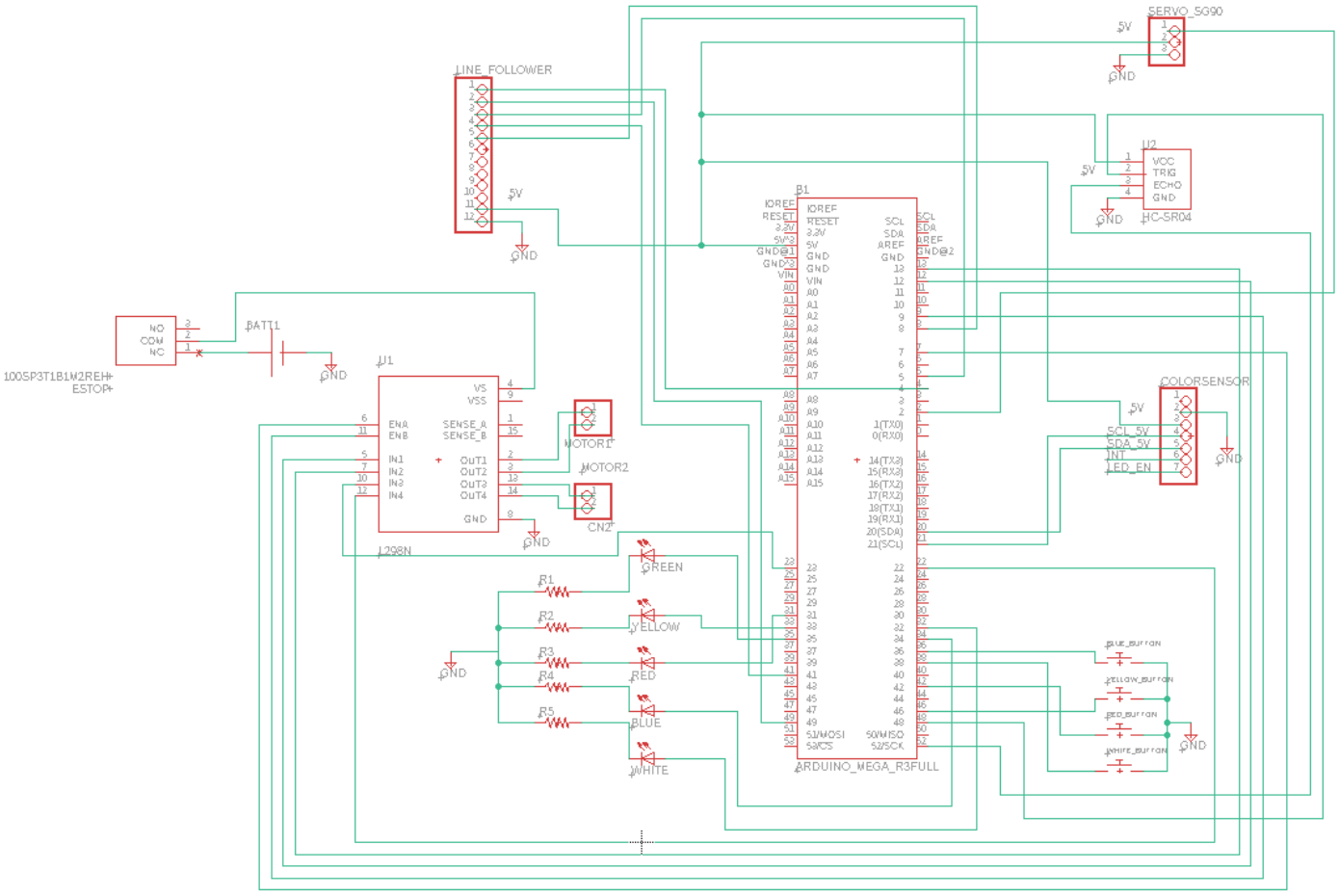
Appendices: CAD, Electrical Circuit, and Snippets of the Code

CAD of Mole- Whacker





Circuit Diagram



Main Code for Robot:

// Color Sensor Pins and Variables

#include <Wire.h>

#include <QTRSensors.h>

#include"Adafruit_TCS34725.h"

Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_101MS, TCS34725_GAIN_4X);

uint8_t colorList[3] = {0, 0, 0};

```
// LED and Button Pins
const int RYB_button[3] = {42, 46, 40};
const int RYB_LED[3] = {31, 33, 34};
const int start_button = 38;
const int start_LED = 32;
const int match_LED = 35;
int color_selected;
// Motor Pins
/*        41  8  5  4
   M1-----Line Follower---------M4  Watch from Here for CW and CCW
   |                       |  CW is forward CCW is backward
   |                       |
   M2--------------------------M3
*/
// Motor 1
const int m1_speed = 9;
const int m1_in1 = 23;
const int m1_in2 = 22;
// Motor 2
const int m2_speed = 3;
const int m2_in1 = 24;
const int m2_in2 = 25;
// Motor 3
```

```
const int m3_speed = 6;

const int m3_in1 = 12;

const int m3_in2 = 13;

// Motor 4

const int m4_speed = 7;

const int m4_in1 = 11;

const int m4_in2 = 10;



// Mole Whacker Servo Pins

#include <Servo.h>

const int servo_pin = 1;

Servo myservo;

int drop_flag = 0; // 0 Means Ball Not Dropped, 1 Means Ball Dropped


// Line Follower Stuff


/*        41  8  5  4

   M1-----Line Follower---------M4  Watch from Here for CW and CCW

   |                    |  CW is forward CCW is backward

   |                    |

   M2--------------------------M3

*/
```

```
QTRSensors qtr;
const uint8_t SensorCount = 4;
uint16_t sensorValues[SensorCount];
int linePassed = 0;

// Ultrasonic Stuff
#define echoPin 52
#define trigPin 48
// defines variables for Ultrasonic Sensor
long int duration; // variable for the duration of sound wave travel
int distance; // variable for the distance measurement
int measured_distance;
int start_flag = 0;


// Global States
int STATE = 1;
const int COLOR_STATE = 1;
const int START_STATE = 2;
const int INITIAL_STATE = 3;
const int FORWARD_STATE = 4;
const int DROP_STATE = 5;
const int END_GAME_STATE = 6;
```

```
/*****************/
void setup() {

  //Ultrasonic Sensor Pin Declarations
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an OUTPUT
  pinMode(echoPin, INPUT); // Sets the echoPin as an INPUT

  // Button Setup
  pinMode(RYB_button[0], INPUT_PULLUP);
  pinMode(RYB_button[1], INPUT_PULLUP);
  pinMode(RYB_button[2], INPUT_PULLUP);

  // Start Setup
  pinMode(start_button, INPUT_PULLUP);

  // LED Setup
  pinMode(RYB_LED[0], OUTPUT);
  pinMode(RYB_LED[1], OUTPUT);
  pinMode(RYB_LED[2], OUTPUT);
  pinMode(start_LED, OUTPUT);
  pinMode(match_LED, OUTPUT);
```

```
// Color Sensor Setup
tcs.begin();
pinMode(RYB_LED[0], OUTPUT);
pinMode(RYB_LED[1], OUTPUT);
pinMode(RYB_LED[2], OUTPUT);

// Motor Setup
pinMode(m1_speed, OUTPUT);
pinMode(m2_speed, OUTPUT);
pinMode(m3_speed, OUTPUT);
pinMode(m4_speed, OUTPUT);

pinMode(m1_in1, OUTPUT);
pinMode(m2_in1, OUTPUT);
pinMode(m3_in1, OUTPUT);
pinMode(m4_in1, OUTPUT);

pinMode(m1_in2, OUTPUT);
pinMode(m2_in2, OUTPUT);
pinMode(m3_in2, OUTPUT);
pinMode(m4_in2, OUTPUT);

// Servo Setup
```

```
  myservo.attach(servo_pin);


  //Line Follower Stuff
  // configure the sensors
  qtr.setTypeRC();
  // qtr.setSensorPins((const uint8_t[]){3, 4, 5, 6, 7, 8, 9, 10}, SensorCount);
  qtr.setSensorPins((const uint8_t[]) {
    4, 5, 8, 41
  }, SensorCount);
  qtr.setEmitterPin(2);


  delay(500);
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, HIGH); // turn on Arduino's LED to indicate we are in calibration
mode


  // 2.5 ms RC read timeout (default) * 10 reads per calibrate() call
  // = ~25 ms per calibrate() call.
  // Call calibrate() 400 times to make calibration take about 10 seconds.
  for (uint16_t i = 0; i < 400; i++)
  {
    qtr.calibrate();
  }
```

```
  digitalWrite(LED_BUILTIN, LOW); // turn off Arduino's LED to indicate we are through with calibration


  // print the calibration minimum values measured when emitters were on
  Serial.begin(9600);
  for (uint8_t i = 0; i < SensorCount; i++)
  {
    Serial.print(qtr.calibrationOn.minimum[i]);
    Serial.print(' ');
  }
  Serial.println();


  // print the calibration maximum values measured when emitters were on
  for (uint8_t i = 0; i < SensorCount; i++)
  {
    Serial.print(qtr.calibrationOn.maximum[i]);
    Serial.print(' ');
  }
  Serial.println();
  Serial.println();
  delay(1000);


}
```

```
// Motor Move Function like forwards right left etc
void forward() {
  int forward_speed = 50;
  Serial.println("Going Straight");
  digitalWrite(m1_in1, LOW); //Clockwise for Motor 1
  digitalWrite(m1_in2, HIGH);
  analogWrite(m1_speed, forward_speed);

  digitalWrite(m2_in2, HIGH); //clockwise for Motor 2
  digitalWrite(m2_in1, LOW);
  analogWrite(m2_speed, forward_speed);

  digitalWrite(m3_in1, HIGH); // clockwise for Motor3
  digitalWrite(m3_in2, LOW);
  analogWrite(m3_speed, forward_speed);

  digitalWrite(m4_in1, LOW); // Clockwise for Motor4
  digitalWrite(m4_in2, HIGH);
  analogWrite(m4_speed, forward_speed);
}

void adjRight() {
```

```
  Serial.println("Shift Right");
  int right_speed = 55;
  digitalWrite(m1_in1, LOW); //Clockwise for Motor 1
  digitalWrite(m1_in2, HIGH);
  analogWrite(m1_speed, right_speed);

  digitalWrite(m2_in2, HIGH); //clockwise for Motor 2
  digitalWrite(m2_in1, LOW);
  analogWrite(m2_speed, right_speed);

  digitalWrite(m3_in1, LOW); // clockwise for Motor3
  digitalWrite(m3_in2, HIGH);
  analogWrite(m3_speed, right_speed);

  digitalWrite(m4_in1, HIGH); // Connections for Motor4
  digitalWrite(m4_in2, LOW);
  analogWrite(m4_speed, right_speed);
}

void adjLeft() {

  Serial.println("Shift Left");
  int left_speed = 60;
```

```
  digitalWrite(m1_in1, HIGH); //Clockwise for Motor 1
  digitalWrite(m1_in2, LOW);
  analogWrite(m1_speed, left_speed);

  digitalWrite(m2_in2, LOW); //clockwise for Motor 2
  digitalWrite(m2_in1, HIGH);
  analogWrite(m2_speed, left_speed);

  digitalWrite(m3_in1, HIGH); // clockwise for Motor3
  digitalWrite(m3_in2, LOW);
  analogWrite(m3_speed, left_speed);

  digitalWrite(m4_in1, LOW); // Connections for Motor4
  digitalWrite(m4_in2, HIGH);
  analogWrite(m4_speed, left_speed);
}
void right() {
  Serial.println("Full Right");
  digitalWrite(m1_in1, LOW); //CCW for Motor 1
  digitalWrite(m1_in2, HIGH);
  analogWrite(m1_speed, 80);

  digitalWrite(m2_in2, HIGH); //CCW for Motor 2
```

```
  digitalWrite(m2_in1, LOW);
  analogWrite(m2_speed, 80);

  digitalWrite(m3_in1, LOW); // CW for Motor3
  digitalWrite(m3_in2, HIGH);
  analogWrite(m3_speed, 80);

  digitalWrite(m4_in1, HIGH); // CW for Motor4
  digitalWrite(m4_in2, LOW);
  analogWrite(m4_speed, 100);
}
void stopMotor() {
  Serial.println("Stop Motor");
  digitalWrite(m1_in1, LOW); //Stop for Motor 1
  digitalWrite(m1_in2, LOW);

  digitalWrite(m2_in2, LOW); //Stop for Motor 2
  digitalWrite(m2_in1, LOW);

  digitalWrite(m3_in1, LOW); // Stop for Motor3
  digitalWrite(m3_in2, LOW);
```

```
  digitalWrite(m4_in1, LOW); // Stop for Motor4
  digitalWrite(m4_in2, LOW);
}
// Ultrasonic Stuff
double UltrasonicSensor()
{
  // Clears the trigPin condition
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin HIGH (ACTIVE) for 10 microseconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // Reads the echoPin, returns the sound wave travel time in microseconds
  duration = pulseIn(echoPin, HIGH);
  // Calculating the distance
  distance = duration * 0.034 / 2; // Speed of sound wave divided by 2 (go and back)
  // Displays the distance on the Serial Monitor
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");
  return distance;
```

```
}


// State Functions
// Max Location of Black Line
int maxValLoc_func(uint16_t sensorValues[]) {
  int maxVal = sensorValues[0];
  int maxValLoc = 0;
  for (int i = 0; i < 3; i++) {
    if (sensorValues[i] > maxVal) {
      maxVal = sensorValues[i];
      maxValLoc = i;
    }
  }
  return maxValLoc;
}
void initial_move() {
  int position;
  if (start_flag == 0) {
    while (sensorValues[2] < 700) {
      position = qtr.readLineBlack(sensorValues);
      forward();
    }
    stopMotor();
```

```
    Serial.println("Start from White Successful");
    start_flag = 1;
    STATE = FORWARD_STATE;
  }
  else {
    STATE = INITIAL_STATE;
  }
}
// Move Forward Function
int move_forward(int color_selected) {
  // read calibrated sensor values and obtain a measure of the line position
  // from 0 to 5000 (for a white line, use readLineWhite() instead)
  int pos_distance[3] = {165, 135, 57};
  int position = qtr.readLineBlack(sensorValues);
  delay(50);
  int maxValLoc = maxValLoc_func(sensorValues);
  Serial.print("Max Val Loc: ");
  Serial.println(maxValLoc);
  for (uint8_t i = 0; i < SensorCount; i++)
  {
    Serial.print(sensorValues[i]);
    Serial.print('\t');
  }
```

```
  // Necessary Corrections to Maintain Straight Line
  // Line in the right
  maxValLoc = maxValLoc_func(sensorValues);
  if (maxValLoc == 0) {
    adjRight();
  }

  // Line at the middle
  else if (maxValLoc == 1) {
    forward();
  }

  // Line at the left
  else if (maxValLoc == 2) {
    adjLeft();
  }
  position = qtr.readLineBlack(sensorValues); // Update Position
  // ULTRASONIC DISTANCE
  measured_distance = UltrasonicSensor();
  if (measured_distance == pos_distance[0] || measured_distance == pos_distance[1] || measured_distance == pos_distance[2]) {
    int lux = tcs.getRGB(&colorList[0], &colorList[1], &colorList[2]);
    if (lux > 700) {
```

```
    if (colorList[0] > 60 && color_selected == 0) {
      digitalWrite(match_LED, HIGH);
      STATE = DROP_STATE;
    }
    else if (colorList[0] > 30 && colorList[1] > 12 && colorList[2] < 5 && color_selected == 1)
{
      digitalWrite(match_LED, HIGH);
      STATE = DROP_STATE;
    }
    else if (colorList[2] > 30 && color_selected == 2) {
      digitalWrite(match_LED, HIGH);
      STATE = DROP_STATE;
    }
    else {
      digitalWrite(match_LED, LOW);
      STATE = FORWARD_STATE;
    }
  }
  else {
    digitalWrite(match_LED, LOW);
    STATE = FORWARD_STATE;
  }
 }
```

```
// First Turn
if (measured_distance < 57 && linePassed < 3) {
  linePassed = linePassed + 1;
  forward();
  delay(100);
  stopMotor();
  delay(500);
  right();
  delay(600);
  position = qtr.readLineBlack(sensorValues); // Update Position
  maxValLoc = maxValLoc_func(sensorValues);
  while (maxValLoc != 0 && sensorValues[0] > 500) {
    right();
    position = qtr.readLineBlack(sensorValues); // Update Position
    maxValLoc = maxValLoc_func(sensorValues);
  }
}
forward();
delay(100);
measured_distance = UltrasonicSensor(); // Update Measured Distance
// Second Turn
if (measured_distance < 120 && linePassed == 3) {
  linePassed = linePassed + 1;
```

```
  forward();
  delay(100);
  stopMotor();
  delay(500);
  right();
  delay(600);
  position = qtr.readLineBlack(sensorValues);
  maxValLoc = maxValLoc_func(sensorValues);
  while (maxValLoc != 0 && sensorValues[0] > 500) {
    right();
    position = qtr.readLineBlack(sensorValues);
    maxValLoc = maxValLoc_func(sensorValues);
  }
  forward();
  delay(100);
}
measured_distance = UltrasonicSensor();
// Third Turn
if (measured_distance < 120 && linePassed == 4) {
  linePassed = linePassed + 1;
  forward();
  delay(100);
  stopMotor();
```

```
  delay(500);
  right();
  delay(600); // Make This 90 Degree Turn
  position = qtr.readLineBlack(sensorValues);
  maxValLoc = maxValLoc_func(sensorValues);
  while (maxValLoc != 0 && sensorValues[0] > 500) {
    right();
    position = qtr.readLineBlack(sensorValues);
    maxValLoc = maxValLoc_func(sensorValues);
  }
  forward();
  delay(100);
 }
if (measured_distance < 120 && linePassed == 5) {
  linePassed = linePassed + 1;
  forward();
  delay(100);
  stopMotor();
  delay(500);
  right();
  delay(600); // Make This 90 Degree Turn
  position = qtr.readLineBlack(sensorValues);
  maxValLoc = maxValLoc_func(sensorValues);
```

```
    while (maxValLoc != 0 && sensorValues[0] > 500) {
      right();
      position = qtr.readLineBlack(sensorValues);
      maxValLoc = maxValLoc_func(sensorValues);
    }
    forward();
    delay(100);
  }
  if (linePassed == 6) {
    STATE = END_GAME_STATE;
  }
  return STATE;
}

int drop_ball() {
  int angle_min = 55;
  int angle_max = 111;
  int pos = 0;    // variable to store the servo position
  //Logic
  delay(1000);
  myservo.write(angle_min);
  for (pos = angle_min; pos < angle_max; pos++) { // goes from 90 to 180 in steps of 15
    delay(50000);
```

```
    myservo.write(pos);
    delay(50);
  }
  return FORWARD_STATE;
}
void loop() {
  // read calibrated sensor values and obtain a measure of the line position
  // from 0 to 5000 (for a white line, use readLineWhite() instead)
  int started;
  switch (STATE) {
    case COLOR_STATE:
      color_selected = 0;
      Serial.println("Color State");
      for (int i = 0; i < 2; i++) {
        if (digitalRead(RYB_button[i]) == LOW) {
          color_selected = i;
          digitalWrite(RYB_LED[i], HIGH);
          STATE = START_STATE;
        }
      }
      break;
    case START_STATE:
      Serial.println("Start State");
```

```
    started = digitalRead(start_button);
    if (started == LOW) {
      digitalWrite(start_LED, HIGH);
      STATE = INITIAL_STATE;
    }
    break;
  case INITIAL_STATE:
    Serial.println("Initial State");
    initial_move();
    STATE = FORWARD_STATE;
  case FORWARD_STATE:
    Serial.println("Forward State");
    STATE = move_forward(color_selected); // Make Function for this
    break;
  case DROP_STATE:
    Serial.println("Drop State");
    STATE = drop_ball();
    break;
  case END_GAME_STATE:
    Serial.println("End Game State");
    stopMotor();
    break;
}
```

```
}
```